

# Streaming $k$ -mismatch with error correcting and applications

Jakub Radoszewski<sup>1</sup> and Tatiana Starikovskaya<sup>2</sup>

<sup>1</sup>University of Warsaw

<sup>2</sup>Université Paris-Diderot

## Abstract

We present a new streaming algorithm for the  $k$ -MISMATCH problem, one of the most basic problems in pattern matching. Given a pattern and a text, the task is to find all substrings of the text that are at the Hamming distance at most  $k$  from the pattern. Our algorithm is enhanced with an important new feature called ERROR CORRECTING, and its complexities for  $k = 1$  and for a general  $k$  match those of the best known solutions for the  $k$ -MISMATCH problem from FOCS 2009 and SODA 2016. As a corollary we develop a series of streaming algorithms for pattern matching on weighted strings, which are a commonly used representation of uncertain sequences in molecular biology.

## 1 Introduction

In this work we design efficient streaming algorithms for a number of problems of approximate pattern matching. In this class of problems we are given a pattern and a text and wish to find all substrings of the text that are “similar” to the pattern. We assume that the text arrives as a stream, one symbol at a time.

The first small-space streaming algorithms for pattern matching were suggested in the pioneering paper by Porat and Porat in FOCS 2009 [18]. In particular, they showed an algorithm for the PATTERN MATCHING problem, where one must find all exact occurrences of the pattern in the text. For a pattern of length  $m$  their algorithm takes only  $\mathcal{O}(\log m)$  space and  $\mathcal{O}(\log m)$  time per each symbol of the text, and reports all exact occurrences of the pattern in the text as they occur. In 2010, Ergun et al. presented a slightly simpler version of the algorithm [13], and finally in 2011 the running time of the algorithm was improved to constant [6]. The next logical step was

to study the complexity of approximate pattern matching in the streaming model in which we are to find all substrings of the text that are at a small distance from the pattern. The most popular distances are the Hamming distance,  $L_1$ ,  $L_2$ ,  $L_\infty$ -distances, and the edit distance. Unfortunately, the general task of computing these distances for each alignment of the pattern and of the text precisely requires at least  $\Omega(m)$  space. This lower bound holds even for randomised algorithms [10].

However, this is not the case if we allow approximation or if it is sufficient to compute the exact distances only when they are small. Here we focus on approximate pattern matching under the Hamming distance. In the  $k$ -MISMATCH problem we are given a pattern and a text, and we must find all substrings of the text that are at the Hamming distance at most  $k$  from the pattern. Let  $m$  be the length of the pattern and  $n$  be the length of the text. If we are interested in computing a  $(1 + \varepsilon)$ -approximation of the Hamming distance for all alignments of the pattern and of the text, then this can be done in  $\mathcal{O}(\varepsilon^{-5} \sqrt{m} \log^4 m)$  space and  $\mathcal{O}(\varepsilon^{-4} \log^3 m)$  time per arriving symbol [11]. And if we are only interested in computing the Hamming distances at the alignments where they do not exceed a given threshold  $k$  (the  $k$ -MISMATCH problem), then Porat and Porat [18] showed a randomised streaming algorithm that solves this problem in  $\mathcal{O}(k^3 \log^7 m / \log \log m)$  space and  $\mathcal{O}(k^2 \log^5 m / \log \log m)$  time per arriving symbol; they also presented an algorithm for a special case of  $k = 1$  with  $\mathcal{O}(\log^4 m / \log \log m)$  space and  $\mathcal{O}(\log^3 m / \log \log m)$  time per arriving symbol. Recently, their result was improved (in terms of the dependency on  $k$ ) to  $\mathcal{O}(k^2 \log^{11} m / \log \log m)$  space and  $\mathcal{O}(\sqrt{k} \log k + \log^5 m)$  time per arriving symbol by Clifford et al. [8].

Our first contribution is a new streaming algorithm for the  $k$ -MISMATCH problem. The crucial feature of our algorithm is that, for each alignment where the Hamming distance is at most  $k$ , it can also output the differences of symbols of the pattern and of the text in the mismatching positions. This is particularly surprising as we are not allowed to store a copy of the pattern or of the text in the streaming setting. The  $k$ -MISMATCH problem extended with computing this additional characteristic is called here the  $k$ -MISMATCH WITH ERROR CORRECTING problem. We first develop a solution for  $k = 1$ .

**Theorem 1.1.** *The 1-MISMATCH WITH ERROR CORRECTING problem can be solved in  $\mathcal{O}(\log^5 m / \log \log m)$  space and  $\mathcal{O}(\log^5 m / \log \log m)$  time per arrival. The probability of error is at most  $1/\text{poly}(n)$ .*

As a corollary we obtain a  $k$ -MISMATCH WITH ERROR CORRECTING

algorithm for arbitrary  $k$  via an existing randomised reduction to the 1-MISMATCH WITH ERROR CORRECTING problem [18, 8].

**Theorem 1.2.** *The  $k$ -MISMATCH WITH ERROR CORRECTING problem can be solved in  $\mathcal{O}(k^2 \log^{10} m / \log \log m)$  space and  $\mathcal{O}(k \log^8 m / \log \log m)$  time per arrival. The probability of error is at most  $1/\text{poly}(m)$ .*

Since in the  $k$ -MISMATCH WITH ERROR CORRECTING problem we need at least  $\Omega(k)$  time per arrival to list the symbol differences, the dependency of the running time of our algorithm on  $k$  is optimal. The time complexity of our  $k$ -MISMATCH WITH ERROR CORRECTING algorithm is better than the time complexity of the  $k$ -MISMATCH algorithm of [18] and worse than that of [8]. Our algorithm also uses less space than the  $k$ -MISMATCH algorithm of [18] (in terms of  $k$ ) and the  $k$ -MISMATCH algorithm of [8]. For the former, it is explained by the fact that we use a more efficient reduction. For the latter, it is because we do not need the elaborated time-saving techniques of Clifford et al. [8] (as the running time of our algorithm is already almost optimal). We note that a similar problem was considered previously in [19] by Porat and Lipsky. They introduced a small-space representation of a stream called a “sketch” that can be efficiently maintained under a symbol change and can be used to compute the Hamming distance between two streams as well as to correct the errors between them. Unfortunately, it is not clear whether their sketches can be efficiently maintained over a sliding window (i.e. for substrings of the text) and therefore we cannot use them in our model.

The ERROR CORRECTING feature is a very powerful tool. We demonstrate this by developing the first streaming algorithms for the problem of pattern matching on weighted strings. A *weighted string* (also known as weighted sequence, position probability matrix or position weight matrix, PWM) is a sequence of probability distributions on the alphabet. Weighted strings are a commonly used representation of uncertain sequences in molecular biology. In particular, they are used to model motifs and have become an important part of many software tools for computational motif discovery, see e.g. [20, 21]. In the WEIGHTED PATTERN MATCHING problem we are given a text and a pattern, both of which can be either weighted or regular strings. If either only the text or only the pattern are weighted, the task is to find all alignments of the text and of the pattern where they match with probability above a given threshold  $1/z$ . In the most general case, when both the pattern and the text are weighted, we must find all alignments of the text and of the pattern where there exists a regular string that matches both the text and the pattern with probability at least  $1/z$ .

As it was already mentioned, we are the first to consider the WEIGHTED PATTERN MATCHING problem in the streaming setting. In the offline setting the most commonly studied variant, when the text is a weighted string of length  $n$  and the pattern is a regular string, can be solved in  $\mathcal{O}(n \log n)$  time via the Fast Fourier Transform [7] or in  $\mathcal{O}(n \log z)$  time using the suffix array and lookahead scoring [16]. This variant has been also considered in the indexing setting, in which we are to preprocess a weighted text to be able to answer queries for multiple patterns; see [1, 14, 5, 3]. The symmetric variant of the WEIGHTED PATTERN MATCHING problem, when only the pattern is weighted, is closely related to the problem of profile matching [17] and admits  $\mathcal{O}(n \log n)$ -time and  $\mathcal{O}(n \log z)$ -time solutions as well. Finally, the variant when both the text and the pattern are weighted was introduced in [4], where an  $\mathcal{O}(nz^2 \log z)$ -time solution was presented. Later a more efficient  $\mathcal{O}(n\sqrt{z} \log \log z)$ -time solution was devised in [16]. The offline algorithms use  $\Omega(m)$  space and the best indexing solution uses  $\Omega(nz)$  space. There has been also considered a problem of computing Hamming and edit distances for weighted strings [2].

We consider each of the three variants of the WEIGHTED PATTERN MATCHING problem. We assume  $z < m$  and for each variant demonstrate two algorithms. The algorithms use different methods to find the alignments where the pattern and the text might match with probability at least  $1/z$ . The second method gives better space complexity for  $\log^{10} m \leq z < m$ . Note that this setting is not uncommon, because even choosing a letter with probability 0.9 at  $\log m$  positions gives match probability  $1/m^c$  for a constant  $c > 0$ . Let  $\mathcal{S}_{\log z} = \mathcal{O}(\log^2 z \cdot \log^{10} m / \log \log m)$  and  $\mathcal{T}_{\log z} = \mathcal{O}(\log z \cdot \log^8 m / \log \log m)$  be the space and the time complexities for the  $k$ -MISMATCH WITH ERROR CORRECTING for  $k = \log z$  and a pattern of length  $m$  (See Theorem 1.2). We first consider the variant when only the pattern is weighted.

**Theorem 1.3.** *If only the pattern is weighted, the WEIGHTED PATTERN MATCHING problem can be solved in  $\mathcal{O}(z \log m)$  space and  $\mathcal{O}(\log \log(z + m))$  time per arrival with error probability  $1/\text{poly}(n)$  or in  $\mathcal{O}(z) + \mathcal{S}_{\log z}$  space and  $\mathcal{T}_{\log z}$  time with error probability  $1/\text{poly}(m)$ .*

For the case when the pattern is a regular string and the text is weighted, we show a  $(1 - \varepsilon)$ -approximation streaming algorithm. At each alignment the algorithm outputs either “Yes” or “No”. If the pattern matches the fragment of the text, the algorithm outputs “Yes”. If the match probability is between  $(1 - \varepsilon)/z$  and  $1/z$ , it can output either “Yes” or “No”, and

otherwise it outputs “No”. If it outputs “Yes”, it also outputs a  $(1 - \varepsilon)$ -approximation of the match probability between  $P$  and  $T$ .

**Theorem 1.4.** *If only the text is weighted, there is a  $(1 - \varepsilon)$ -approximation streaming algorithm that solves the WEIGHTED PATTERN MATCHING problem in  $\mathcal{O}(z \log_{1/(1-\varepsilon)} z + z \log m)$  space and  $\mathcal{O}(z \log z)$  time per arrival with error probability  $1/\text{poly}(n)$  or in  $\mathcal{O}(z \log_{1/(1-\varepsilon)} z) + \mathcal{S}_{\log z}$  space and  $\mathcal{O}(z \log z) + \mathcal{T}_{\log z}$  time with error probability  $1/\text{poly}(m)$ .*

Our final result is a  $(1 - \varepsilon)$ -approximation streaming algorithm for the most general case when both the pattern and the text are weighted. At each alignment the algorithm outputs either “Yes” or “No”. If there is a regular string that matches the text and the pattern at this alignment with probability above  $1/z$ , the algorithm outputs “Yes”. It may also output “Yes” if there is a string that matches with probability between  $(1 - \varepsilon)/z$  and  $1/z$ . Otherwise it outputs “No”.

**Theorem 1.5.** *If both the text and the pattern are weighted, there is a  $(1 - \varepsilon)$ -approximation streaming algorithm that solves the WEIGHTED PATTERN MATCHING problem in  $\mathcal{O}(z \log_{1/(1-\varepsilon)} z + z^2 \log m)$  space and  $\mathcal{O}(z \log z + z \log \log(z + m))$  time per arrival with error probability  $1/\text{poly}(n)$  or in  $\mathcal{O}(z \log_{1/(1-\varepsilon)} z) + z \cdot \mathcal{S}_{\log z}$  space and  $\mathcal{O}(z \log z) + z \cdot \mathcal{T}_{\log z}$  time per arrival with error probability  $1/\text{poly}(m)$ .*

**Model of computation.** We account for all the space we use, and cannot afford to store a copy of the text or of the pattern. We assume that we receive the pattern first and can preprocess it by reading it in a streaming fashion several times. After having preprocessed the pattern we receive the text that arrives as a stream, one symbol at a time. The indicated error probabilities are per arrival. We assume a constant-sized alphabet  $\Sigma$ . A symbol of a weighted string is a vector of probabilities of the letters in which all entries are of the form  $c^{p/2^{dw}}$ , where  $w$  is the machine word size,  $c$  and  $d$  are constants, and  $p$  is an integer that fits in a constant number of machine words (log-probability model). Additionally, the probability 0 has a special representation. The only operations on probabilities in our algorithms are multiplications and divisions, which can be performed exactly in  $\mathcal{O}(1)$  time in this model.

## 2 Weighted Pattern Matching

In this section we present our solutions to the three variants of the WEIGHTED PATTERN MATCHING problem. The solution to the  $k$ -MISMATCH WITH ERROR CORRECTING problem is deferred until Section 3.

### 2.1 Proof of Theorem 1.3 — only pattern is weighted

We start by giving an algorithm that takes  $\mathcal{O}(z \log m)$  space and  $\mathcal{O}(\log \log(z + m))$  time per arrival. At the preprocessing step we generate all (regular) strings that match the pattern with probability at least  $1/z$ . We do this in a naive fashion by maintaining a set of regular strings that match the prefix of the pattern received so far with probability at least  $1/z$ . When a new position of the pattern arrives, we try to extend each string in the set with all possible letters and update the match probabilities. Note that the size of the set never exceeds  $z$ , as the sum of the probabilities over all strings in the set is at most 1. If a substring of the text  $T$  matches  $P$ , it must be an occurrence of one of the generated strings. We find all such occurrences using the streaming DICTIONARY MATCHING algorithm by Clifford et al. [9] for the dictionary of the  $z$  generated strings. For a dictionary of  $z$  strings of length  $m$  the algorithm takes  $\mathcal{O}(z \log m)$  space and  $\mathcal{O}(\log \log(z + m))$  time per symbol. The algorithm is randomised and has error probability  $1/\text{poly}(n)$ . This gives us the first algorithm of Theorem 1.3.

To explain the second algorithm, we define  $\mathcal{H}(P)$  to be a regular string obtained from  $P$  by choosing at each position the symbol with the maximum probability. At each alignment where  $P$  and  $T$  match with probability at least  $1/z$ , the number of mismatches between  $\mathcal{H}(P)$  and  $T$  is at most  $\log z$  (because at each mismatch the probability of  $P$  and  $T$  to match is at most  $1/2$ ). To find such alignments, we use our  $k$ -MISMATCH algorithm for  $k = \log z$ . It remains to explain how we retrieve the match probabilities. We only need to retrieve the match probabilities if they are  $\geq 1/z$ , i.e. the  $m$ -length substring of the text must match exactly one of the at most  $z$  strings generated in the first solution. Let  $M$  be the set of all different mismatches between the generated strings and  $\mathcal{H}(P)$ ; each mismatch is specified by its position  $i$  and a letter  $a \neq \mathcal{H}(P)[i]$ . We claim that  $|M| \leq z$ . Indeed, from each mismatch  $(i, a)$  from  $M$  one can produce a regular string that matches  $P$  with probability  $\geq 1/z$  by taking  $\mathcal{H}(P)$  and replacing  $\mathcal{H}(P)[i]$  by  $a$ , and there are at most  $z$  such strings.

When reading the pattern, we compute a set  $M' \supseteq M$  of mismatches  $(i, a)$  together with the values  $p(i, a) = \Pr[P[i] = a] / \Pr[P[i] = \mathcal{H}(P)[i]]$ . We

insert all pairs  $(i, a)$  for  $i = 1, \dots, m$ ,  $a \neq \mathcal{H}(P)[i]$  into two balanced BSTs, one indexed by  $p(i, a)$  and the other by  $(i, a)$ . We keep the size of  $M'$  not greater than  $z$  by removing the elements with the smallest  $p(i, a)$  if needed. At the end, using the second balanced BST, we can retrieve the match probability of the candidate occurrence and  $P$  in  $\mathcal{O}(\log^2 z) = \mathcal{O}(\log z \log m)$  time ( $z < m$ ). Thus we arrive at the conclusion of Theorem 1.3.

## 2.2 Proof of Theorem 1.4 — only text is weighted

In this section we show two solutions to the WEIGHTED PATTERN MATCHING problem for a regular pattern  $P$  and a weighted text  $T$ . We assume that  $\varepsilon < 1/2$ . If  $\varepsilon > 1/2$ , we can use the  $1/2$ -approximation algorithm that has the same asymptotic complexity and better approximation factor. We start by giving a definition of maximal matching suffixes of prefixes of the text  $T$  that will play a crucial role in both algorithms.

**Definition 2.1.** *A maximal matching suffix of  $T[1, q]$  is a (regular) string  $S$  such that  $S$  matches  $T[q - |S| + 1, q]$  with probability  $\geq 1/(2z)$  and either  $|S| = q$  or any string  $aS$ , for  $a \in \Sigma$ , matches  $T[q - |S|, q]$  with probability  $< 1/(2z)$ .*

The reason for selecting the cut-off value of  $1/(2z)$  will become clear later (in Lemma 2.2, where we need the cut-off to be at least  $(1 - \varepsilon)/z$ ). The number of maximal matching suffixes of  $T[1, q]$  never exceeds  $2z$  [1]. Note that  $P$  matches  $T$  at an alignment  $q$  with probability  $\geq 1/z$  if and only if there is a maximal matching suffix  $S$  of  $T[1, q]$  that ends with it. Moreover, the match probability between  $P$  and  $T$  is equal to the match probability between the  $m$ -length suffix of  $S$  and  $T$ .

We maintain each maximal matching suffix  $S$  and the probabilities  $x_i$  of a match between  $S$  and  $T[i]$  as a stream. Let  $\mathcal{H}(T)$  be a regular string obtained from  $T$  by choosing at each position the symbol with the maximum probability. We call a position  $i$  in the stream a *mismatch position* if the corresponding letter in  $S$  is different than  $\mathcal{H}(T)[i]$ ; note that we then have  $x_i \leq 1/2$ . Let  $r$  be the rightmost mismatch position in the stream such that  $\prod_{i \geq r+1} x_i \leq 1/(2z)$ . If such mismatch position does not exist, we put  $r = 0$ . Note that there are at most  $\log z + 1$  mismatch positions to the right of  $r$ . We index the stream by these mismatch positions and the differences between the letters of  $S$  and  $\mathcal{H}(T)$  in these positions. We also store the total product of numbers located to the right of each of these at most  $\log z + 1$  mismatch positions.

Let  $\Sigma_{q+1}$  be the set of letters  $b \in \Sigma$  such that  $\Pr[T[q+1] = b] \geq 1/(2z)$ . At the arrival of position  $q+1$  we create  $|\Sigma_{q+1}|$  copies of each stream. We then add  $\Pr[T[q+1] = b]$  for each  $b \in \Sigma_{q+1}$  to the  $b$ -th copy of each stream and update the mismatch positions and all related information in  $\mathcal{O}(\log z)$  time in a straightforward manner. At this moment the index  $r$  in some streams might have increased and we might have more than one stream for a single maximal matching suffix. On the other hand, the streams that correspond to a fixed maximal matching suffix have the same indices. We can therefore delete the duplicates as follows. The first step is to build a trie (see [12] for a definition) on the indices of the streams in  $\mathcal{O}(z \log z)$  time and space to sort the streams. The indices are inserted into the trie from right to left. (Children in the trie are stored using perfect hashing.) The second step is to select one stream for each index and delete the remaining ones.

We now explain how we maintain an approximate value of the match probability between the  $m$ -length suffix of  $S$  and  $T$ . To this end, we introduce the SLIDING WINDOW PRODUCT problem. In the SLIDING WINDOW PRODUCT problem we are given a stream of numbers from  $[0, 1]$  arriving one number at a time, an integer  $m$ , and a threshold  $1/z$ , and must find the product of numbers in each window of length  $m$  provided that it is above the threshold. We give a  $(1 - \varepsilon)$ -approximation algorithm for this problem. More precisely, when a new symbol arrives, the algorithm outputs either a number or “No”. If it outputs a number, it is a  $(1 - \varepsilon)$ -approximation of the product of the last  $m$  numbers, and otherwise the product is at most  $(1 - \varepsilon)/z$ .

**Lemma 2.2.** *For a stream of numbers  $\{x_i\}_{i=1}^\infty$ , where  $x_i \in [0, 1]$ , arriving one at a time, and a window width  $m$ , there is a deterministic  $(1 - \varepsilon)$ -approximation algorithm that takes  $\mathcal{O}(\log_{1/(1-\varepsilon)} z)$  space and  $\mathcal{O}(1)$  time per arrival and solves the SLIDING WINDOW PRODUCT problem.*

*Proof.* At time  $q$  the algorithm maintains a family of at most  $M(z) = \lfloor 2 \log_{1-\varepsilon}((1 - \varepsilon)/z) \rfloor$  intervals  $[i_1, i_2 - 1], [i_2, i_3 - 1], \dots, [i_{k(q)}, q]$ , such that  $1 \leq i_1 < i_2 < \dots < i_{k(q)} < i_{k(q)+1} = q + 1$ , which obeys the following invariant.

- (i) All intervals  $[i_j, i_{j+1} - 1]$  for  $j \geq 2$  are subintervals of  $[q - m + 1, q]$ , and  $[i_1, i_2 - 1]$  has a non-empty intersection with  $[q - m + 1, q]$ ;
- (ii) If  $x_{i_j} < 1 - \varepsilon$ , then  $i_{j+1} = i_j + 1$ ;



- (iii) Otherwise,  $x_{i_j} \cdot x_{i_j+1} \cdot \dots \cdot x_{i_{j+1}-1} \geq 1 - \varepsilon$  and either  $j = k(q)$  or  $x_{i_j} \cdot x_{i_j+1} \cdot \dots \cdot x_{i_{j+1}} < 1 - \varepsilon$ .

The algorithm stores the product of numbers in each interval and the total product of numbers in all intervals. When  $x_{q+1}$  arrives, it updates the family in the following way. Let  $\pi$  be the product of numbers in the interval  $[i_{k(q)}, q]$ . If  $\pi \cdot x_{q+1} \geq 1 - \varepsilon$ , then the algorithm extends the interval  $[i_{k(q)}, q]$  by the element  $x_{q+1}$ . Otherwise, it creates a new interval  $[q+1, q+1]$ . If the number of intervals becomes larger than  $M(z)$  or if  $i_2 \leq (q+1) - m + 1$ , the algorithm deletes the leftmost interval  $[i_1, i_2 - 1]$ . Finally it updates the total product of numbers in the intervals. Let us now explain how the algorithm exploits the intervals.

Note that the product of numbers in each two consecutive intervals is at most  $1 - \varepsilon$ . Therefore, if  $q - m + 1 < i_1$ , then the product of the last  $m$  numbers is at most  $(1 - \varepsilon)/z$  and the algorithm outputs “No”. Otherwise from the invariant it follows that  $q - m + 1 \in [i_1, i_2 - 1]$ . In this case we output the total product of the numbers in the intervals. Recall that if  $[i_1, i_2 - 1]$  is not a singleton interval, then the product of numbers in it is at least  $1 - \varepsilon$ . Therefore, the answer will be a  $(1 - \varepsilon)$ -approximation of  $x_{q-m+1} \cdot x_{q-m+2} \cdot \dots \cdot x_q$ . The space complexity follows from the fact that  $M(z) = \lfloor 2 \log_{1-\varepsilon}((1 - \varepsilon)/z) \rfloor$ .  $\square$

Finally, we explain how we find a maximal matching suffix that ends with  $P$ . We will give two different methods resulting in the two algorithms of Theorem 2.2. The first method is based on a straightforward application of the streaming PATTERN MATCHING algorithm [6]. This algorithm uses  $\mathcal{O}(\log m)$  space and takes  $\mathcal{O}(1)$  time to process a text symbol. In the first method, we simply run the algorithm for the pattern  $P$  and each of the maximal matching suffixes streams. The second method is based on our  $k$ -MISMATCH algorithm. If  $P$  matches  $T$  at some alignment  $q$  with probability at least  $1/z$ , then  $\mathcal{H}(T)[q - m + 1, q]$  is a  $\log z$ -mismatch occurrence of  $P$  (see Section 2.1). We use our  $k$ -MISMATCH algorithm with  $k = \log z$  for  $P$  and  $\mathcal{H}(T)$  to find all such alignments. When we identify a  $\log z$ -mismatch occurrence of  $P$  in  $\mathcal{H}(T)$ , we find a stream that corresponds to a maximal matching suffix that ends with  $P$ , if any (using the indexing trie). Suppose we have found a maximal matching suffix  $S$  that ends with  $P$  using either of the two methods. We then use the algorithm of Lemma 2.2 to compute a  $(1 - \varepsilon)$ -approximation of the product of the last  $m$  probabilities in the stream associated with  $S$  and output it as an answer.

Let us analyse the two algorithms. Since the number of maximal match-

ing suffixes never exceeds  $2z$ , the total number of streams is  $\mathcal{O}(z)$ . Updating the streams, including the duplicate removal, takes  $\mathcal{O}(z \log z)$  space and time per position. For each of the streams we run the algorithm of Lemma 2.2, which takes  $\mathcal{O}(z \log_{1/(1-\varepsilon)} z)$  space and  $\mathcal{O}(z)$  time per position. Therefore, the first algorithm requires  $\mathcal{O}(z \log_{1/(1-\varepsilon)} z + z \log m)$  space and  $\mathcal{O}(z \log z)$  time per position. The error probability of the PATTERN MATCHING algorithm and, therefore, of our algorithm is  $1/\text{poly}(n)$  per position. The second algorithm can output an incorrect answer only when the  $k$ -MISMATCH algorithm errs, which happens with probability  $1/\text{poly}(m)$ . For  $k = \log z$ , the  $k$ -MISMATCH algorithm takes  $\mathcal{S}_{\log z}$  space and  $\mathcal{T}_{\log z}$  time per arriving symbol. Finally, to find the “right” stream we need just  $\mathcal{O}(\log z)$  additional time per position. In total, this is  $\mathcal{O}(z \log_{1/(1-\varepsilon)} z) + \mathcal{S}_{\log z}$  space and  $\mathcal{O}(z \log z) + \mathcal{T}_{\log z}$  time per position.

### 2.3 Proof of Theorem 1.5 — both the text and the pattern are weighted

We conclude by showing a streaming algorithm for the WEIGHTED PATTERN MATCHING problem when both the pattern and the text are weighted. Our solution combines the ideas of the two previous sections. Recall that at each alignment the algorithm must output “Yes” if there is a regular string that matches the text and the pattern with probability above  $1/z$ . The algorithm may also output “Yes” if there is a string that matches with probability between  $(1 - \varepsilon)/z$  and  $1/z$ . Otherwise it outputs “No”.

For the pattern  $P$  we generate the at most  $z$  regular strings that match it with probability  $\geq 1/z$ . For the text  $T$  we maintain a set of at most  $2z$  streams for the maximal matching suffixes in the text. For each of the streams we run the algorithm of Lemma 2.2. There is a regular string that matches the pattern and the text with probability  $\geq 1/z$  if and only if the following two conditions hold: (i) one of the maximal matching suffixes ends with a string generated for the pattern and (ii) the product of the last  $m$  probabilities in the suffix’s stream is at least  $1/z$ . We can verify the second condition using the approximation of Lemma 2.2 as in Section 2.2. To verify the first condition, we again give two methods. The first method is to run the DICTIONARY MATCHING algorithm for the strings generated for the pattern and each of the maximal matching suffixes. The second method is based on our  $k$ -MISMATCH algorithm. If  $P$  matches  $T$  at some alignment  $q$ , then  $\mathcal{H}(T)[q - m + 1, q]$  must be a  $\log z$ -occurrence of one of the strings we generated for the pattern. We use our  $k$ -MISMATCH algorithm with  $k = \log z$  for each of the generated strings and  $\mathcal{H}(T)$  to find all such occurrences. At

each alignment  $q$  we consider  $\log z$ -occurrences of the generated strings and build a trie of their indices, where the indices are defined as in Section 2.2. If a maximal matching suffix ends with one of these strings, its index must be in the trie. We can check this condition is  $\mathcal{O}(z \log z)$  time in total.

Let us analyse the two algorithms. Updating the suffix streams takes  $\mathcal{O}(z \log z)$  time as there are  $\mathcal{O}(z)$  of them. For each of the streams we run the algorithm of Lemma 2.2, which takes  $\mathcal{O}(z \log_{1/(1-\varepsilon)} z)$  space and  $\mathcal{O}(z)$  time. Hence, the first method takes  $\mathcal{O}(z \log_{1/(1-\varepsilon)} z + z^2 \log m)$  space and  $\mathcal{O}(z \log z + z \log \log(z + m))$  time per position. The error probability is  $1/\text{poly}(n)$ . The second method can output an incorrect answer only when one of the  $k$ -MISMATCH algorithms errs, which happens with probability  $z/\text{poly}(m) = 1/\text{poly}(m)$ . For  $k = \log z$  the  $k$ -MISMATCH algorithm takes  $\mathcal{S}_{\log z}$  space and  $\mathcal{T}_{\log z}$  time per symbol. Building the trie of mismatch encodings takes  $\mathcal{O}(z \log z)$  time. Searching for the indices in the trie takes  $\mathcal{O}(z \log z)$  time. In total, this is  $\mathcal{O}(z \log_{1/(1-\varepsilon)} z) + z \cdot \mathcal{S}_{\log z}$  space and  $\mathcal{O}(z \log z) + z \cdot \mathcal{T}_{\log z}$  time.

### 3 $k$ -mismatch with error correcting

In this section we give our solution to the  $k$ -MISMATCH problem for  $k = 1$  (Theorem 1.1) and for a general value of  $k$  (Theorem 1.2).

#### 3.1 Proof of Theorem 1.1: $k = 1$

Consider two  $m$ -length strings  $X$  and  $Y$ . In [18] Porat and Porat showed that the Hamming distance between  $X$  and  $Y$  is equal to 1 if and only if for each prime  $q \in [\log m, 2 \log m]$  there is exactly one  $j \leq q$  such that subsequences  $X[j]X[j+q]X[j+2q]\dots$  and  $Y[j]Y[j+q]Y[j+2q]\dots$  are not equal. This observation formed the foundation of their 1-MISMATCH algorithm [18]. They also noticed that, by the Chinese remainder theorem, the starting positions of the non-equal subsequences uniquely determine the mismatch position between  $X$  and  $Y$ . However, their algorithm discards all the information about the mismatching subsequences as soon as it discovers a mismatch and, therefore, cannot be augmented with the ERROR CORRECTING feature. To overcome this difficulty, we introduce a notion of 1-mismatch sketches.

Let us start by recalling the definition of a rolling hash function called Rabin-Karp fingerprint [15] and its properties. The Rabin-Karp fingerprint of a string  $X = X[1] \dots X[\ell]$  is defined as  $\phi(X) = (\sum_{i=1}^{\ell} X[i] \cdot r^i) \bmod p$ , where  $p$  is a prime and  $r$  is a random integer in  $\mathbb{F}_p$ . Importantly, if we

choose  $p$  to be large enough, then the collision probability of any two  $\ell$ -length strings  $X$  and  $Y$ , where  $\ell \leq m$ , will be at most  $1/\text{poly}(m)$ . We will also need the following lemma which is a straightforward corollary of the definition of Rabin-Karp fingerprints.

**Lemma 3.1.** *Let  $X, Y$  be two strings,  $Z = XY$  be their concatenation, and let  $\varphi(X)$ ,  $\varphi(Y)$ , and  $\varphi(Z)$  be the Rabin-Karp fingerprints of  $X$ ,  $Y$ , and  $Z$ , respectively. Then*

$$\varphi(Z) = (\varphi(X) + r^{|X|} \cdot \varphi(Y)) \bmod p.$$

Therefore if we assume that together with a Rabin-Karp fingerprint of a string of length  $\ell$  we store  $r^\ell \bmod p$  and  $r^{-\ell} \bmod p$ , then knowing the Rabin-Karp fingerprints of two of the strings  $X$ ,  $Y$ ,  $Z$ , one can compute the Rabin-Karp fingerprint of the third string in  $\mathcal{O}(1)$  time. We are now ready to give the definition of 1-mismatch sketches.

**Definition 3.2** (1-mismatch sketch). *For a prime  $q$ , the 1-mismatch sketch of a string  $X$  is a vector of length  $q$ , where the  $j$ -th element is the Rabin-Karp fingerprint of the subsequence  $X[j]X[j+q]X[j+2q]\dots$*

**Lemma 3.3.** *Let  $X, Y$  be two strings,  $Z = XY$  be their concatenation. Consider the 1-mismatch sketches of  $X$ ,  $Y$ , and  $Z$  defined for a prime  $q$ . Then:*

- (i) *If  $X = Y$ , then their 1-mismatch sketches are equal;*
- (ii) *Given the 1-mismatch sketches of  $X$  and  $Y$ , we can compute the 1-mismatch sketch of  $Z$  in  $\mathcal{O}(q)$  time;*
- (iii) *Given the 1-mismatch sketches of  $X$  and  $Z$ , we can compute the 1-mismatch sketch of  $Y$  in  $\mathcal{O}(q)$  time;*
- (iv) *Given the 1-mismatch sketch of  $X$ , we can compute the 1-mismatch sketch of  $X^\alpha$  in  $\mathcal{O}(q \log \alpha)$  time, where  $\alpha$  is an integer and  $X^\alpha$  is a concatenation of  $\alpha$  copies of  $X$ .*

*Proof.* Property (i) is a direct corollary of the definitions of Rabin-Karp fingerprints and 1-mismatch sketches. As for Property (ii), note that we need to compute the Rabin-Karp fingerprints of the at most  $q$  concatenations of pairs of strings, similarly for Property (iii) we only need to compute the Rabin-Karp fingerprints of the at most  $q$  strings constructed from  $Y$  given the Rabin-Karp fingerprints of the at most  $q$  strings constructed from  $X$  and their concatenations (in  $Z$ ). Finally, Property (iv) follows from Property (ii) as we can compute the 1-mismatch sketch of a square of any string given its 1-mismatch sketch in  $\mathcal{O}(q)$  time.  $\square$

**Overview of the solution.** We reduce the 1-MISMATCH WITH ERROR CORRECTING problem to  $\mathcal{O}(\log m)$  instances of a special case of this problem where the mismatch is required to belong to the *second half* of the pattern. More formally, consider  $\lceil \log m \rceil + 1$  partitions  $P = P_i S_i$ , where  $P_i$  is a prefix of length  $\min\{2^i, m\}$  and  $S_i$  is the remaining suffix, for  $i = 0, 1, \dots, \lceil \log m \rceil$ . We say that a substring of  $T$  is a *right-half 1-mismatch occurrence* of  $P_i$  if either  $i = 0$  and  $P_0$  does not match, or  $i \geq 1$  and the mismatch is at position  $j > 2^{i-1}$  in  $P_i$ .

**Observation 3.4.** *Any 1-mismatch occurrence of  $P$  in  $T$  is a right-half 1-mismatch occurrence of some  $P_i$  followed by an exact occurrence of  $S_i$ .*

We run two parallel processes for each  $i$ . The first process searches for right-half 1-mismatch occurrences of  $P_i$ . After having found a right-half 1-mismatch occurrence, it passes the information about it (including its 1-mismatch sketches) to the second process that decides if it is followed by an exact occurrence of  $S_i$ . Let us recall a streaming PATTERN MATCHING algorithm that lies in the foundation of both processes. For a pattern  $Q$  and text  $T$  the algorithm takes  $\mathcal{O}(\log |Q|)$  space and  $\mathcal{O}(\log |Q|)$  per symbol. This algorithm is not the best known solution, but it gives the desired time bounds and is conceptually simple. The algorithm stores  $\mathcal{O}(\log |Q|)$  levels of positions of  $T$ . Positions in level  $j$  are occurrences of  $Q[1, 2^j]$  in the suffix of the current text  $T$  of length  $2^{j+1}$ . The algorithm stores the Rabin-Karp fingerprints of  $T$  from the beginning of  $T$  up to each of these positions. If there are at least 3 such positions at one level, then there is a large overlap of the occurrences of  $Q[1, 2^j]$  and therefore all the positions form a single arithmetic progression whose difference equals the length of the minimal period of  $Q[1, 2^j]$ . This allows to store the aforementioned information very compactly, using only  $\mathcal{O}(\log |Q|)$  space in total. Finally, the algorithm stores the Rabin-Karp fingerprint of the current text and of all prefixes  $Q[1, 2^j]$ . When a new symbol  $T[q]$  arrives, the algorithm considers the leftmost position  $\ell_j$  in each level  $j$ . If  $q - \ell_j + 1$  is smaller than  $2^{j+1}$ , the algorithm does nothing. Otherwise if the fingerprints imply that  $\ell_j$  is an occurrence of  $Q[1, 2^{j+1}]$ , the algorithm promotes it to the next level, and if  $\ell_j$  is not an occurrence of  $Q[1, 2^{j+1}]$ , the algorithm discards it. When a position reaches the top level, it is an occurrence of  $Q$  and the algorithm outputs it. For more details, see [18, 13, 6].

**1-mismatch occurrences of  $P_i$ .** Suppose a new symbol  $T[q]$  has arrived. To check whether  $T[q - |P_i| + 1, q]$  is a right-half 1-mismatch occurrence of  $P_i$ ,

we use the 1-MISMATCH algorithm [18], which requires  $\mathcal{O}(\log^4 m / \log \log m)$  space and  $\mathcal{O}(\log^3 m / \log \log m)$  time per symbol. This algorithm also returns the mismatch position. However, the algorithm does not know neither the difference of symbols at the mismatch position nor the 1-mismatch sketches of the occurrence. To extend the algorithm with the required functionality, we use the PATTERN MATCHING algorithm for a pattern  $Q = P_i$  and  $T$ . If  $T[q - |P_i| + 1, q]$  is a right-half 1-mismatch occurrence of  $P_i$ , then  $P_i[1, 2^{i-1}]$  matches at the position  $q - |P_i| + 1$  exactly. It follows that at time  $q$  the position  $q - |P_i| + 1$  is stored at level  $i - 1$  of the PATTERN MATCHING algorithm, and we know the Rabin-Karp fingerprints of  $T[1, q - |P_i|]$  and  $T[1, q]$ . Therefore, we can compute the Rabin-Karp fingerprint of  $T[q - |P_i| + 1, q]$  in  $\mathcal{O}(1)$  time. Let  $j$  be the mismatch position and  $\phi(P_i)$  and  $\phi(T[q - |P_i| + 1, q])$  be the Rabin-Karp fingerprints of  $P_i$  and  $T[q - |P_i| + 1, q]$ , respectively. We use the fingerprints to compute the difference of symbols of  $P_i$  and  $T$  at the position  $j$ .

**Lemma 3.5.** *Assume that  $X$  and  $Y$  are two strings of length  $m$  that differ only at position  $j$ . Knowing the Rabin-Karp fingerprints of  $X$  and  $Y$ , one can compute  $X[j] - Y[j]$  in  $\mathcal{O}(\log m)$  time.*

*Proof.* Let  $\phi(X)$  and  $\phi(Y)$  be the Rabin-Karp fingerprints of  $X$  and  $Y$ , respectively. Then  $X[j] - Y[j]$  is equal to  $(\phi(X) - \phi(Y)) \cdot r^{-j} \pmod{p}$ , where  $p$  and  $r$  are the integers used in the definition of Rabin-Karp fingerprints. This formula can be evaluated in  $\mathcal{O}(\log m)$  time.  $\square$

As a final step we compute the 1-mismatch sketches of  $T[q - |P_i| + 1, q]$  and send them to the second process. We assume that during the preprocessing step the algorithm precomputes the 1-mismatch sketches of  $P_i$  for all primes in  $[\log m, 2 \log m]$ , which requires  $\mathcal{O}(\log^2 m / \log \log m)$  space. It can then restore the 1-mismatch sketches of  $T[q - |P_i| + 1, q]$  in  $\mathcal{O}(\log^2 m / \log \log m)$  time by fixing the symbol in the mismatch position.

**Exact occurrences of  $S_i$ .** We now describe the second process built on top of the PATTERN MATCHING algorithm for the pattern  $Q = S_i$  and  $T$ . Since for each new position  $q$  the first process tells whether it is preceded by a right-half 1-mismatch occurrence of  $P_i$ , all we need is to carry this information from the level 0 of the PATTERN MATCHING algorithm to the top level. We claim that it suffices to store the 1-mismatch sketches for a constant number of positions in each level. Using them, we will be able to infer the remaining unstored information.

Consider level  $j$ . The progression that the algorithm currently stores for this level can be a part of a longer progression of occurrences of  $S_i[1, 2^j]$  in  $T$ . Let  $\ell$  be some occurrence of  $S_i[1, 2^j]$  for which we would like to figure out whether it is preceded by a right-half 1-mismatch occurrence of  $P_i$ . If  $\ell$  is far from the start of the long progression, then the text preceding  $\ell$  is periodic with period  $\rho_j$  being the shortest period of  $S_i[1, 2^j]$ , and we can use this fact to infer the 1-mismatch information. So our main concern is the positions  $\ell$  that are at the distance of at most  $|P_i| = 2^i$  from the start of the long progression. We define four positions  $\ell_j^a$ ,  $a = 1, 2, 3, 4$ , that help us to restore the information in this case. Note that we can easily determine the moment when a new long progression starts, as this is precisely the moment when the difference between two consecutive positions in level  $j$  becomes larger than  $\rho_j$ . We define  $\ell_j^1$  as the first position preceded by a right-half 1-mismatch occurrence of  $P_i$  that was added to level  $j$  since that moment. We further define  $\ell_j^a$  as the leftmost terms preceded by a right-half 1-mismatch occurrence of  $P_i$  in  $[\ell_j^1 + (a-1) \cdot 2^{i-2}, \ell_j^1 + a \cdot 2^{i-2}]$  for  $a = 2, 3, 4$ . (If such terms exist; otherwise they are left undefined). The algorithm stores the 1-mismatch sketches of  $T[1, \ell_j^a - 2^i - 1]$  for each  $a = 1, 2, 3, 4$  and the 1-mismatch sketches of  $T[1, \ell_j^1 - 1]$ .

Besides, the algorithm stores a number of 1-mismatch sketches for the pattern, which are computed during the preprocessing step and occupy  $\mathcal{O}(\log^3 m / \log \log m)$  space in total. First, it stores the 1-mismatch sketches of the minimal period of  $P_i[1, 2^{i-1}]$ . Second, the algorithm stores the 1-mismatch sketches of the minimal period  $S_i[1, \rho_j]$  of  $S_i[1, 2^j]$  for each  $j = 0, 1, \dots, \lfloor \log |S_i| \rfloor$ . Finally, it stores the 1-mismatch sketches of  $S_i[1, \delta_j]$ , where  $\delta_j$  is the remainder of  $-2^i$  modulo  $\rho_j$ . To compute the periods and the sketches during the preprocessing step we make three passes over the pattern. Over the first two passes we compute the minimal periods of  $P_i[1, 2^{i-1}]$  and  $S_i[1, 2^j]$  for all  $i$  and  $j$  using  $\mathcal{O}(\log^2 m)$  instances of the streaming algorithm [13] (the algorithm can compute the minimal period in one pass if it is small, but requires two passes if the period is large). During the third pass we compute the 1-mismatch sketches.

**Lemma 3.6.** *The algorithm can decide if a position  $\ell$  in level  $j$  is preceded by a right-half 1-mismatch occurrence of  $P_i$  in  $\mathcal{O}(\log^3 m / \log \log m)$  time and, if this is the case, it can also compute the 1-mismatch sketches of  $T[1, \ell - 2^i - 1]$  and  $T[1, \ell - 1]$ .*

*Proof.* We consider three cases based on the relationship between  $\ell$  and the positions  $\ell_j^a$ .

**Case 1:**  $\ell < \ell_j^1$  or  $\ell_j^a + 2^{i-2} \leq \ell < \ell_j^{a+1}$  for some  $a \in \{1, 2, 3\}$ . In this case  $\ell$  cannot be preceded by a 1-mismatch occurrence of  $P_i$  by definition.

**Case 2:**  $\ell_j^a \leq \ell < \ell_j^a + 2^{i-2}$ . Below we explain how to compute the 1-mismatch sketches of  $T[\ell - 2^i, \ell - 1]$  in  $\mathcal{O}(\log^3 m / \log \log m)$  time. The algorithm can then use these sketches and the ones of  $P_i$  to determine in  $\mathcal{O}(\log^2 m / \log \log m)$  time whether  $\ell$  is preceded by a right-half 1-mismatch occurrence of  $P_i$ .

First, the algorithm computes the 1-mismatch sketches of  $T[1, \ell - 2^i - 1]$  by extensively using Lemma 3.3. Recall that  $\ell_j^a$  is preceded by a right-half 1-mismatch occurrence of  $P_i$ . If the same property holds for  $\ell$ , then  $T[\ell_j^a - 2^i, \ell_j^a - 2^{i-1} - 1]$  and  $T[\ell - 2^i, \ell - 2^{i-1} - 1]$  are two occurrences of  $P_i[1, 2^{i-1}]$  that overlap by at least  $2^{i-2}$  positions; see Fig. 1. Therefore  $T[\ell_j^a - 2^i, \ell - 2^i - 1]$  is a power of the minimal period of  $P_i[1, 2^{i-1}]$  by Fine and Wilf's periodicity lemma [12]. The algorithm stores the 1-mismatch sketches of  $T[1, \ell_j^a - 2^i - 1]$  and it can compute the 1-mismatch sketches of  $T[\ell_j^a - 2^i, \ell - 2^i - 1]$  from the 1-mismatch sketches of the minimal period of  $P_i[1, 2^{i-1}]$  in  $\mathcal{O}(\log^3 m / \log \log m)$  time. Therefore, it can compute the 1-mismatch sketches of  $T[1, \ell - 2^i - 1]$  in  $\mathcal{O}(\log^3 m / \log \log m)$  time.

Second, the algorithm computes the 1-mismatch sketches of  $T[1, \ell - 1]$  using the 1-mismatch sketches of  $T[1, \ell_j - 1]$  and of the minimal period of  $S_i[1, 2^j]$  in  $\mathcal{O}(\log^3 m / \log \log m)$  time. Finally, the algorithm computes the 1-mismatch sketches of  $T[\ell - 2^i, \ell - 1]$  in  $\mathcal{O}(\log^2 m / \log \log m)$  time using the 1-mismatch sketches of  $T[1, \ell - 2^i - 1]$  and  $T[1, \ell - 1]$ .

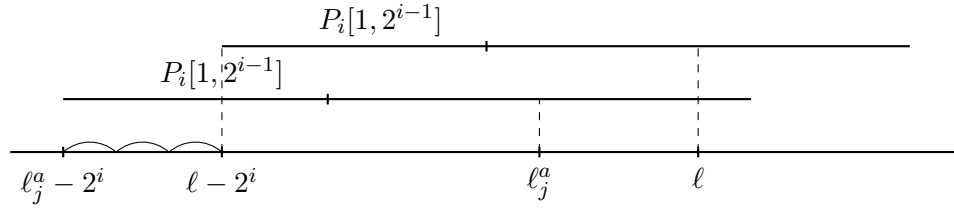


Figure 1: Case 2 of Lemma 3.6. Position  $\ell$  is close to one of the positions  $\ell_j^a$ .

**Case 3:**  $\ell \geq \ell_j^1 + 2^i$ . In this case  $T[\ell - 2^i, \ell - 1]$  is a suffix of  $T[\ell_j^1, \ell - 1]$ , which is a power of the minimal period of  $S_i[1, 2^j]$ , that is,  $S_i[1, \rho_j]$ . As we know the 1-mismatch sketches of  $T[1, \ell_j^1 - 1]$ ,  $S_i[1, \rho_j]$ , and  $S_i[1, \delta_j]$  (recall



that  $\delta_j$  is the remainder of  $-2^i$  modulo  $\rho_j$ ), the algorithm can use Lemma 3.3 to compute the 1-mismatch sketches of  $T[1, \ell - 1]$  (by extending  $T[1, \ell_j^1 - 1]$  with a power of  $S_i[1, \rho_j]$ ) and  $T[\ell - 2^i, \ell - 2^i - 1]$  (by subtracting the sketches of  $S_i[1, \delta_j]$  from the sketches of a power of  $S_i[1, \rho_j]$ ) in  $\mathcal{O}(\log^3 m / \log \log m)$  time. It can then use them to determine whether  $T[\ell - 2^i, \ell - 1]$  is a 1-mismatch occurrence of  $P_i$ .  $\square$

The algorithm uses the lemma both to compile the output and to update the levels. When the algorithm encounters a position in the top level that is preceded by a right-half 1-mismatch occurrence of  $P_i$  and followed by an exact occurrence of  $S_i$ , the algorithm outputs it together with the required difference of symbols, which is computed from the 1-mismatch sketches using Lemma 3.5. Let us show how the algorithm updates the levels. When a new symbol  $T[q]$  arrives and  $T[q] = S_i[1]$ , the algorithm adds  $q$  to the level 0. If  $q$  is preceded by a right-half 1-mismatch occurrence of  $P_i$ , then the algorithm also tries to update the  $\ell_0^a$  values and possibly retains the 1-mismatch sketches of  $T[1, q - 2^i - 1]$  and  $T[1, q - 1]$  output by the first process. The algorithm then updates each of the remaining levels in turn. To update level  $j$ , it considers the leftmost position  $\ell_j$  in this level. If the Rabin-Karp fingerprints imply that it is an occurrence of  $S_i[1, 2^{j+1}]$ , the algorithm promotes it to the next level, and otherwise discards it. In the former case the algorithm uses Lemma 3.6 to check whether  $\ell_j$  is preceded by a right-half 1-mismatch occurrence of  $P_i$ . If it does, it computes the 1-mismatch sketches and updates the positions  $\ell_{j+1}^a$ .

**Complexity.** Recall that for each  $i = 0, 1, \dots, \lceil \log m \rceil$  we run two processes in parallel. The bottleneck of the first process is the 1-MISMATCH algorithm; it uses  $\mathcal{O}(\log^4 m / \log \log m)$  space and  $\mathcal{O}(\log^3 m / \log \log m)$  time per symbol. The time complexity of the second process is bounded by  $\mathcal{O}(\log m)$  applications of the algorithm of Lemma 3.6 (one per level) and is  $\mathcal{O}(\log^4 m / \log \log m)$  per symbol. The space complexity of the second process is  $\mathcal{O}(\log^3 m / \log \log m)$ . Therefore, our 1-MISMATCH WITH ERROR CORRECTING algorithm uses  $\mathcal{O}(\log^5 m / \log \log m)$  space and time per symbol.

### 3.2 Proof of Theorem 1.2: general value of $k$

Theorem 1.2 follows from Theorem 1.1 via a randomised reduction. The first variant of this reduction, which was deterministic, was presented by Porat and Porat [18], who used it to reduce the  $k$ -MISMATCH problem to

the 1-MISMATCH problem. It was further made more space-efficient at a return of slightly higher error probability by Clifford et al. [8]. The main idea of this reduction is to consider a number of partitions of the pattern into  $\mathcal{O}(k \log^2 m)$  subpatterns. By defining the partitions appropriately, we can guarantee that at each alignment where the Hamming distance is small, each mismatch will correspond to a 1-mismatch occurrence of some subpattern. This lets us apply the 1-MISMATCH WITH ERROR CORRECTING algorithm to find all such alignments and to restore the data for them. We give a full description of the reduction below.

We start by filtering out the locations where the Hamming distance is large. To this end, we use a randomised algorithm for the following 2-approximate  $k$ -mismatch problem. Let  $H$  be the true Hamming distance at a particular alignment of the pattern and the text. The algorithm outputs “Yes” if  $H \leq k$ , either “Yes” or “No” if  $k < H \leq 2k$ , and “No” if  $H > 2k$ .

**Lemma 3.7** ([8]). *Given a pattern  $P$  of length  $m$  and a streaming text arriving one symbol at a time, there is a randomised  $\mathcal{O}(k^2 \log^6 m)$ -space algorithm which takes  $\mathcal{O}(\log^5 m)$  worst-case time per arriving symbol and solves the 2-approximate  $k$ -mismatch problem. The probability of error is at most  $1/m^2$ .*

Next, we consider  $\ell = \lfloor \log m \rfloor$  partitions of the pattern into equispaced subpatterns. More precisely, we select  $\ell$  random primes from the interval  $[k \log^2 m, 34k \log^2 m]$ . For each prime  $p_i$  the pattern  $P$  is then partitioned into  $p_i$  subpatterns  $P_{i,j} = P[j]P[p_i + j]P[2p_i + j] \dots$ , where  $j = 1, \dots, p_i$ . Consider a particular location  $q$  in the text and the set of all mismatches between  $P$  and  $T[q - m + 1, q]$ . Let us call a mismatch *isolated* if it is the only mismatch between some subpattern  $P_{i,j}$  and the corresponding subsequence of  $T[q - m + 1, q]$ , and let  $M_q$  be the set of all isolated mismatches at an alignment  $q$ .

**Lemma 3.8** ([8]). *If  $|M_q| \leq 2k$ , the set  $M_q$  contains all mismatches between  $P$  and  $T[q - m + 1, q]$  with probability at least  $1 - 1/m^2$ .*

To finalize the reduction, we partition the text  $T$  into  $p_i$  equispaced substreams  $T_{i,j}$ , where  $j = 1, \dots, p_i$ , for each prime  $p_i$ . We run the 1-MISMATCH WITH ERROR CORRECTING algorithm for all  $\sum_i p_i^2 = \mathcal{O}(k^2 \log^5 m)$  subpattern/substream pairs  $(P_{i,j_1}, T_{i,j_2})$ . At each location where the algorithm for the 2-approximate problem outputs “Yes”, we retrieve all isolated mismatches and the data using the appropriate  $p_i$  instances of the 1-MISMATCH WITH ERROR CORRECTING problem for each  $i$ . This completes the de-

scription of our solution to the  $k$ -MISMATCH WITH ERROR CORRECTING problem.

In total, we use  $\mathcal{O}(k^2 \log^{10} m / \log \log m)$  space. To analyse the time complexity, note that when a new symbol of  $T$  arrives, we need to send it to  $\mathcal{O}(\log m)$  substreams only (one for each prime) and run the next step for each of the  $\mathcal{O}(k \log^2 m)$  instances of the 1-MISMATCH WITH ERROR CORRECTING problem on each of these substreams, which requires  $\mathcal{O}(k \log^8 m / \log \log m)$  time per symbol.

## References

- [1] Amihood Amir, Eran Chencinski, Costas S. Iliopoulos, Tsvi Kopelowitz, and Hui Zhang. Property matching and weighted matching. *Theor. Comput. Sci.*, 395(2-3):298–310, 2008.
- [2] Amihood Amir, Costas Iliopoulos, Oren Kapah, and Ely Porat. Approximate matching in weighted sequences. In *CPM’06: Proc. of the 17th Annual Symposium on Combinatorial Pattern Matching*, volume 4009 of *LNCS*, pages 365–376, 2006.
- [3] Carl Barton, Tomasz Kociumaka, Solon P. Pissis, and Jakub Radoszewski. Efficient index for weighted sequences. In *CPM’16: Proc. of the 27th Annual Symposium on Combinatorial Pattern Matching*, LIPIcs, pages 4:1–4:13, 2016.
- [4] Carl Barton and Solon P. Pissis. Linear-time computation of prefix table for weighted strings. In *WORDS’15: Proc. of the 10th International Conference on Combinatorics on Words*, volume 9304 of *LNCS*, pages 73–84, 2015.
- [5] Sudip Biswas, Manish Patil, Sharma V. Thankachan, and Rahul Shah. Probabilistic threshold indexing for uncertain strings. In *EDBT’16: Proc. of the 19th International Conference on Extending Database Technology*, pages 401–412, 2016.
- [6] Dany Breslauer and Zvi Galil. Real-time streaming string-matching. In *CPM’11: Proc. of the 22nd Annual Symposium on Combinatorial Pattern Matching*, volume 6661 of *LNCS*, pages 162–172, 2011.
- [7] Manolis Christodoulakis, Costas S. Iliopoulos, Laurent Mouchard, and Kostas Tsichlas. Pattern matching on weighted sequences. In *Comp-BioNets’04: Proc. of the 1st International Conference on Algorithms*

and Computational Methods for Biochemical and Evolutionary Networks, KCL publications, 2004.

- [8] Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana Starikovskaya. The  $k$ -mismatch problem revisited. In *SODA'16: Proc. of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2039–2052, 2016.
- [9] Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana A. Starikovskaya. Dictionary matching in a stream. In *ESA'15: Proc. of the 23rd Annual European Symposium on Algorithms*, volume 9294 of *LNCS*, pages 361–372, 2015.
- [10] Raphaël Clifford, Markus Jalsenius, Ely Porat, and Benjamin Sach. Space lower bounds for online pattern matching. *Theor. Comput. Sci.*, 483:58–74, 2013.
- [11] Raphaël Clifford and Tatiana Starikovskaya. Approximate Hamming distance in a stream. In *ICALP'16: Proc. of the 43rd International Colloquium on Automata, Languages, and Programming*, LIPIcs, pages 20:1–20:13, 2016.
- [12] Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on Strings*. Cambridge University Press, 2007.
- [13] Funda Ergün, Hossein Jowhari, and Mert Saglam. Periodicity in streams. In *APPROX'10: Proc. of the 13th International Workshop on Approximation, Randomization, and Combinatorial Optimization*, volume 6302 of *LNCS*, pages 545–559, 2010.
- [14] Costas S. Iliopoulos, Christos Makris, Yannis Panagis, Katerina Perdikuri, Evangelos Theodoridis, and Athanasios K. Tsakalidis. The weighted suffix tree: An efficient data structure for handling molecular weighted sequences and its applications. *Fundam. Inform.*, 71(2-3):259–277, 2006.
- [15] Richard M. Karp and Michael O. Rabin. Efficient randomized pattern-matching algorithms. *IBM J. of Research and Development*, 31(2):249–260, 1987.
- [16] Tomasz Kociumaka, Solon P. Pissis, and Jakub Radoszewski. Pattern matching and consensus problems on weighted sequences and profiles. In *Proceedings of the 27th International Symposium on Algorithms and Computation, ISAAC 2016. To appear*, 2016.

- [17] Cinzia Pizzi and Esko Ukkonen. Fast profile matching algorithms - A survey. *Theor. Comput. Sci.*, 395(2-3):137–157, 2008.
- [18] Benny Porat and Ely Porat. Exact and approximate pattern matching in the streaming model. In *FOCS'09: Proc. of the 50th Annual Symposium on Foundations of Computer Science*, pages 315–323, 2009.
- [19] Ely Porat and Ohad Lipsky. Improved sketching of hamming distance with error correcting. In *CPM'07: Proc. of the 18th Annual Conference on Combinatorial Pattern Matching*, volume 4580 of *LNCS*, pages 173–182, 2007.
- [20] Albin Sandelin, Wynand Alkema, Pär Engström, Wyeth W. Wasserman, and Boris Lenhard. JASPAR: an open-access database for eukaryotic transcription factor binding profiles. *Nucl. Acids Res.*, 32(1):D91–D94, 2004.
- [21] Xuhua Xia. Position weight matrix, Gibbs sampler, and the associated significance tests in motif characterization and prediction. *Scientifica*, 2012. Article ID 917540.